A Framework for Representing and Manipulating Tonal Music

Steven Abrams, Robert Fuhrer, Daniel V. Oppenheim, Don P. Pazel, James Wright
{abrams, rfuhrer, music, pazel, jwright @watson.ibm.com }
Computer Music Center
IBM T. J. Watson Research Center
P.O. Box 218, Yorktown Heights, NY 10598

Abstract

This paper describes a novel set of tools, collectively known as "Smart Harmony", that allows the composer to control and manipulate music at a high level, while constraining the music to conform to a designated functional tonal framework. The Smart Harmony system permits two essential types of operations. First, it allows musical material to be taken from one context and used in a new context – that is, with a new key and/or chord progression – and automatically transforms the material to sound appropriate in that context. Second, it enables musical material to be reshaped, pulling notes higher or lower in pitch, while constraining the material to sound appropriate within a given harmony. An important benefit of our mechanism derives from its relaxation of a strict interpretation of harmonic function, by which means we gain the flexibility to perform these manipulations. We describe data structures and algorithms to implement this mechanism, and offer motivation steeped in traditional functional harmony for its behavior. Lastly, we present musical examples.

1. INTRODUCTION

The goal of our research in music composition systems is to better support the composer's creative workflow. In order to more fully understand composers' creative needs, we have engaged in extensive dialogs with many composers from different backgrounds. We have concluded from these dialogs that most systems fail to provide adequate mechanisms to: (1) capture a composer's musical ideas quickly in a variety of forms, (2) manipulate them in musically sensible and interesting ways, and (3) organize those ideas in meaningful ways.

More specifically, composers have expressed their need to quickly explore musical territory by reusing and reshaping musical materials within a tonal context. To that end, we have found a useful set of pitch-related musical manipulations that is not supported by current tools. This paper discusses one such family of musically interesting manipulations that are aimed at addressing this need, as embodied in our "Smart Harmony" system.

The Smart Harmony conceptual framework was first designed by Oppenheim and prototyped in DMIX (Oppenheim: 1996), loosely based on a representation of Pachet (1993]) with the addition of functional elements of tonality. The current Smart Harmony implementation, described here, utilizes new data structures and algorithms that build on what we learned from the DMIX prototype. It is a central aspect of our Music Sketcher technology preview (Abrams et. al.:1999), which can be freely downloaded from http://www.research.ibm.com/music.

Smart Harmony embodies a novel representation of functional harmony that models the functional role of pitches within the context of a tonal framework that consists of a harmonic progression and a tonal scale.

The Smart Harmony system supports two basic manipulations:

1) Changing the chord progression associated with musical material, and automatically altering each pitch so that it retains aspects of its harmonic function in the new context.

2) Shifting musical pitches higher or lower, preserving aspects of their harmonic function.

Because these transformations preserve certain aspects of the harmonic function of the individual notes, the overall result is that the altered pitches sound "appropriate".

These two operations offer a variety of useful and musically sensible possibilities, including:

- changing the shape of a musical line of pitches by means of a time-varying curve, maintaining aspects of harmonic function
- reusing musical material from one section or piece of music in another context, adapting it to conform to the new context

Preservation of harmonic function is a fundamentally unique aspect of Smart Harmony. In order to support this, a pitch's function in a particular context is explicitly indicated in its representation. Although our current representation does not capture harmonic function in its entirety, it enables the specific transformations mentioned above and provides room for future growth as well.

In fact, there is a tension between *preserving* the harmonic function, register, and basic melodic shape of musical materials as they are transformed, and allowing the composer *flexibility* to change chord progressions, shift materials in register, and re-shape melodies. Smart

Harmony attempts to strike a balance among these difficult constraints by preserving certain aspects of harmonic function while giving the composer some flexibility to explore the musical territory.

The remainder of this paper discusses the design and implementation of the Smart Harmony system, as embodied in the Music Sketcher technology preview. It begins with a discussion of harmonic function, emphasizing those aspects of harmonic function that are preserved by our transformations. Next, it relates the transformations of Smart Harmony to traditional functional harmony. It goes on to discuss the representation of pitches and chords, as well as the algorithms for performing the basic manipulations. Practical examples are given for each of these transformations. Finally, we present an agenda for enhancing the Smart Harmony system in a number of interesting directions.

2. PRESERVING HARMONIC FUNCTION

The crux of the problem is: How can we perform the musical transformations described above in such a way that the music retains its basic character and still sounds appropriate? The difficult questions are defining what we mean by "retains its basic character" and what makes it "sound appropriate?" Let us examine this by starting with a very simple transformation.

If the only difference between the source and target harmonies is the tonic of the key, then a simple chromatic transposition will clearly preserve the harmonic function of all pitches in the sample. For example, if the original harmony is the simple I-IV-V-I chord progression in C major, and the target harmony is the same progression in F major, a transposition by a perfect fourth will meet the constraint.

However, what if the target harmony is a completely different progression in a different key? If a simple chromatic or diatonic transposition is used, pitches that are harmonious in the original harmony may now be inharmonious in the target and vice-versa. In other words, the result would likely not sound appropriate.

The Smart Harmony system attempts to solve this problem by modeling the relationship between each pitch and the underlying chord and scale in context, and then using that relationship to determine what pitch to use when a note is placed into a new context. It is this relationship – primarily, whether or not the pitch is a (possibly altered) chord-tone or not and, if not, whether it is a diatonic scale-tone or not – that the smart harmony system preserves.

To satisfy this, we represent the number of scale-steps a pitch is from the root of the chord, along with its alteration (if it is a chromatically altered note). We refer to this as the pitch's *modal-degree.* This is distinct from its *scale-degree,* i.e. the number of scale-steps from the tonic of the scale, which is not explicitly represented in our system (but can trivially and automatically be derived knowing the key and root of the chord). Representing the modal degree makes it easy to retain the relationship between a pitch and the local harmony, as embodied by the chord. When a note is moved to a new harmonic context, its modal degree will be used in conjunction with the new chord/scale combination to find the appropriate pitch.

For certain chord changes, this is not sufficient, as there are modal degrees that are chord-tones in some chords, but not in others. For example, the F in a Csus4 chord is modal degree 4, and a chord-tone. If this note were moved to a C major chord, modal degree 4 would now be a non-chord tone. Clearly, this would not be appropriate, as a previously harmonious tone is now inharmonious.

To solve this problem, we allow notes to have one modal-degree, but function as another. That is, the F in the Csus4 example really functions as the third in that chord.[1] Therefore, we additionally represent the *functional modal degree* of a pitch for chord-tones (for non chord-tones, the functional modal degree is not utilized). The F in the Csus4 has functional modal degree 3, and, when moved to a C major triad, would become an E. Representing both the modal and functional modal degrees allows what we term the "preservation of harmonic function."

As mentioned earlier, Smart Harmony supports more than just a chord-change operation – it also gives the composer the ability to raise or lower pitches by a specified amount, while preserving the harmonic function. We call this a "pitch-shift."

To allow pitch-shifting while retaining the basic character of the music, we found it necessary to relax the constraint that preserves the precise relationship between a note and its underlying chord and scale. To solve this, we created the notion of "functional compatibility," which allows a note (as determined by its modal degree and alteration) to be shifted to any other similarly altered but compatible modal degree. The pitch-shift operation first transposes a note by a given amount of semitones, and then "snaps" it to the nearest pitch having a compatible harmonic function with that of the original note in its original context.

In the current incarnation of Smart Harmony, two pitches are considered compatible if they are both chord-tones, or both non chord-tones, and are similarly altered. In other words, using the C major chord example, C, E, and G are all compatible with one another, and D, F, A, and B, are likewise compatible with one another. To illustrate the behavior of alterations, note that Db, Fb, Ab, and Bb are all compatible with one another, as are D#, F#, and A# and B#. [2]

[1] Currently, the functional modal degree of each chord pitch is defined in the chord object, thereby avoiding any ambiguity.

[2] Note that B# is compatible with altered non-chord tones, despite being enharmonically equivalent to C, a chord-tone. Smart Harmony views names as being indicative of the intended function and therefore treats B# as a raised non-chord tone. It

By following the chord-change operation with a specially computed pitch-shift operation, Smart Harmony produces pitches that are not only functionally compatible with their original counterparts, but are also as close as possible to the original pitch. This is accomplished by computing the semitone distance from the original pitch to the result of the chord-change, and pitch-shifting by this amount in the opposite direction of the move. This does not result in the precise original pitch, but rather a nearby compatible pitch. We call this the "preservation of register."

Although "functional compatibility" dilutes the proper notion of harmonic function, without it, the shift operation would be impossible. This is part of the trade-off between a rigid adherence to the rules of functional harmony and the flexibility afforded by our transformations. Nevertheless, the definition of functional compatibility used in Smart Harmony preserves certain important aspects of harmonic function – at least the harmonicity (or lack thereof) of the notes are preserved, for example. The following section provides more details on the relationship between our notions of preserving function and functional compatibility and traditional functional harmony.

Functional Harmony and Smart Harmony

To summarize, we have outlined techniques for preserving harmonic function and register, and defined a notion of functional compatibility. We have shown how these ideas combine to form constraints on a set of musical transformations – changes of chord and pitch shifts. In the following, we examine these constraints and transformations from the perspective of traditional functional harmony. We begin by relating this discussion to the roots of the tonal system: stable and dynamic tones. (In the following discussion we adopt the terminology of (Sadai:1980); similar notions are found in numerous sources.

According to functional harmony, scale degrees 1, 3, and 5 are perceived as being stable, whereas degrees 2, 4, 6, and 7 are perceived as dynamic. The latter form a tonal tension that is resolved by moving to the nearest stable tone. Degrees 2 and 4 can resolve by a step up or down (down is considered the more stable resolution giving a greater sensation of finality).

The relationship between stable and dynamic tones is perhaps the central idea of tonal music. Melodies in the tonal system are built from a small set of fixed *melodic patterns* that either extend or connect neighboring stable tones through prolongation and elaboration (Sadai (1980) mentions six such patterns, see p. 8, and example pp. 18-19; Schenker (1969) demonstrates their application on the macro structure). The rules of traditional voice leading, counterpoint, and harmony all take great care to properly resolve the dynamic tones. Sensations of tension and relaxation, as well as expectation and surprise, may be controlled through careful consideration of these relationships. These sensations have been associated with emotion and meaning in music (Mayer 1969), and are also important in delineating musical phrases and structure. In short, there is little doubt of the importance of the role of stable and dynamic tones – as determined by their scale-degrees – in functional harmony.

The Smart Harmony system, however, deals with modal degrees, not scale-degrees. Moreover, the functional compatibility computation preserves the roles of chord-tones and non chord-tones, not the stability referred to in traditional harmony texts. What, then, is the relationship between stable and dynamic tones in traditional harmony and Smart Harmony's notions of preserving harmonic function and functional compatibility?

In order to support transformations with the desired flexibility, we chose to consider each chord in the underlying harmony as governing a local region, creating locally stable and dynamic tones based on their function in the chord. This borrows from the notion of tonal areas, in which a scale degree becomes a temporary tonic. Our model considers that something similar is happening at the micro-level during each chord – a notion that is widely recognized in jazz theory, for example (Levine:1995). In effect, this representation allows the flexibility of changing chord-progressions – even changing the harmonic rhythm – while preserving a sense of the *local* stability and dynamism of each note, relative to the local chord. The important question of *global* stability and dynamism is left to the composer in his choice of a harmonic progression.

When considering melodic patterns in terms of locally stable and dynamic tones, Smart Harmony's retention of functional compatibility will ensure that every melodic pattern in a source phrase will map onto some melodic pattern in the result. This is a necessary first step, but in many cases, there is room for improvement. The new patterns should also be functionally correct, i.e. unstable tones should function correctly in relation with each other and with the stable tones that delineate each new pattern.

Not surprisingly, this implies that the ideal Smart Harmony system should consider proper voice-leading and resolution of dynamic tones. The degree to which this is important in practice depends a great deal on the musical context in which the transformation is made. For example, one can expect that an inner voice within a rich texture, a melody line that is the listener's focus, and a bass line will each require different treatment.

Another benefit of preserving stability with respect to the local chord rather than the underlying scale is evident when the system is used outside of the domain of classical music. In many later forms of music – jazz, to be sure, but late romantic music or Debussy as well – the dynamic tones of the scale are utilized for added color and therefore need no resolution. 9th, 11th, and 13th chords are the norm rather than the exception – even in a final cadence. Modeling these pitches as chord-tones in a more

would be simple to address this issue with small changes to our current algorithm.

complex chord, rather than as non chord-tones in a simple triad or 7th chord, better models their true function. This also allows the composer to control the degree and nature of added harmonic color by changing the chord-progression, and Smart Harmony will automatically utilize – or drop – the upper partials of more complicated chords.

3. IMPLEMENTATION

Given the notions of modal-degree and functional modal-degree defined above, we can now proceed to explain the exact representations of pitch and harmony used for in Smart Harmony. Underlying Smart Harmony is a set of basic musical objects, implemented in C++, such as named pitches, intervals, and the like, along with appropriate computations; these will not be discussed, for the sake of brevity. The focus will be on the objects unique to the Smart Harmony system.

Representations of Harmonic Objects

There are two basic objects represented in Smart Harmony – a *harmonic pitch*, and a *harmonic context.* The harmonic pitch combines a MIDI note number with a description of its harmonic function (i.e. whether or not the pitch is a chord-tone, a scale-tone, or a chromatic or diatonic alteration of a specified sort from the underlying chord or scale). The harmonic pitch is shown in figure 1:

```
struct HarmonicPitch
{
        int originalMIDIPitch;
        int modalDegree;
        int functionalModalDegree;
        int alteration;
};
```

Figure 1: Harmonic Pitch data structure

As discussed above, the modal degree represents the number of scale-steps from the root of the current chord, and the alteration is a signed integer representing semitone alteration from that pitch (+1 for sharp, -1 for flat). They combine to represent the interval from the root of the chord.

The functionalModalDegree field is only (and always) utilized for chord-tones; for non chord-tones, it is zero. It is used to represent pitches that are actually at one modal degree, but function as a different modal degree, such as the 4 in a sus4 chord, in the earlier example.

The originalMIDIPitch field represents the actual MIDI pitch of the note as it existed in its original harmonic context. It is not relevant for most operations, as it has no functional significance. It does, however, provide a reference point for operations that attempt to minimize the change in pitch that results from the harmonic manipulations, as will be discussed below.

The modal degree, functional modal degree, and alteration fields only partially specify a pitch. Ultimately, to be rendered, the pitch requires a harmonic context.

A harmonic context consists of chord and tonality objects. The Chord Object indicates the scale-degree of the chord root (i.e. its Chord Degree), and the Chord Type. A Chord Type consists of the list of Chord Pitches in the chord, each represented as its interval from the chord root and a description of its harmonic function. The Tonality indicates the key and scale, represented by a {tonic, modality} pair. The tonic is either a MIDI pitch or a named pitch, and the modality is a representation of the intervals in the underlying scale. The entire structure of a harmonic context is shown in figure 2.

The representation of the Chord Pitches is certainly noteworthy. It consists of three numbers: The semitone distance from the root of the chord, its modal degree, and its functional modal degree. As described above, this allows, for example, the fourth in a sus4 chord to be indicated as functionally equivalent to a third.

Looking at figure 2, we see that a Csus4 in the key of F is represented as a V sus4 in F major. The sus4 itself shows the three notes of the chord – the root, the fourth, and the fifth. The root of the chord is (trivially) zero semitones distant from the root of the chord, and is both actually and functionally the first modal degree. It is therefore represented as {0, 1, 1}. The fourth is five semitones from the root, has modal degree 4, but, being a sus4, functions as a third. Hence, this pitch is represented as {5, 4, 3}. The fifth is represented as {7, 5, 5}.

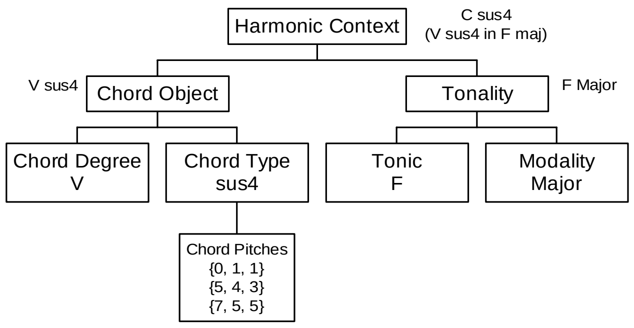

Figure 2: Harmonic Context

Rendering and Transformation Algorithms

The chord representation does not directly yield the pitches in the chord, as named pitches or as MIDI pitches. However, these are easily computed from the Chord Pitches, Chord Degree, and Tonality shown in figure 2.

The computation of a harmonic pitch from a MIDI pitch is more interesting. This computation is central to Smart Harmony, because it turns basic MIDI data plus a standard harmonic analysis into harmonic pitches, amenable to further manipulation by Smart Harmony.

This computation can be ambiguous for chromatically altered notes. Since the harmonic pitch contains both modal-degree and semitone fields, there are times that the system needs to guess which of two enharmonic representations should be used for a note. For example, if a G# (MIDI pitch 68) is played over a C major triad, the system can interpret this as a raised fifth or a lowered sixth. In the current implementation, this decision is made by the underlying Tonality. That allows the G# to be interpreted one way in when the C Major is a I in C Major, and a second way when it is a IV in F Major.

Computing a harmonic pitch from a MIDI pitch is shown in figure 9. First, we explicitly check that the exact pitch is not actually in the chord. This is necessary in cases where the chord contains chromatically altered pitches with respect to the underlying scale, where an incorrect enharmonic choice might result in misclassifying chord-tone as a non-chord tone.

Next, we compute the scale-degree of the pitch (handling enharmonics as described above). Then, we compute the modal-degree as the distance from the chord-degree to the scale-degree. If any notes of the chord have the computed modal-degree, then the pitch is a chord-tone, with a functional modal degree obtained from that of the appropriate note of the chord. Finally, the appropriate alterations are computed.

Another central algorithm of Smart Harmony is the computation of a MIDI pitch from a harmonic pitch in a given harmonic context, shown in figure 7. This operation is invoked whenever a harmonic pitch is moved to a new harmonic context as part of the chord-change transformation. The idea is simple: start from the tonic of the scale, move up by the appropriate number of semi-tones to the chord root, and then by the appropriate number of semitones from the chord root to the correct pitch in the chord. Certain details (mostly related to keeping the pitch in the desired octave) are glossed over in the pseudo-code, but the basic algorithm should be clear.

As we mentioned earlier, one of the operations supported is a pitch-shift operation, whereby a pitch is moved higher or lower by an approximate amount, retaining its harmonic function. This is implemented by the addition of a pitch shift to the computation of a MIDI pitch from a harmonic pitch. The system uses slightly different strategies for chord-tones than for non-chord tones, but both rely on the notion of functional equivalence. Pseudo-code for this algorithm is shown in figure 8.

First, the `MIDIPitchFromHarmonicPitch` routine is used to compute a starting point. Then, a target MIDI pitch is computed by raising this pitch exactly the desired amount of semitones. Then, the closest compatible pitch is found. Essentially, all compatible pitches within the octave are computed, and the closest one is selected. When the harmonic pitch being shifted is a chord-tone, each chord-tone is computed (considering any alterations on the input harmonic pitch), and the closest one to the target pitch is chosen. Likewise, for non chord-tones, each similarly altered non chord-tone is computed, and the closest one is chosen. This implementation supports the definition of functional compatibility offered earlier.

4. EXAMPLES

Figure 3 presents an example of applying several Smart Harmony transformations in Music Sketcher to the first four bars of the melody of Mozart's Piano Sonata in C Major. To import the melody into Music Sketcher, it was manually annotated with its chord progression. The progression was then used to automatically derive the sequence of harmonic pitch objects as described in Section 3.

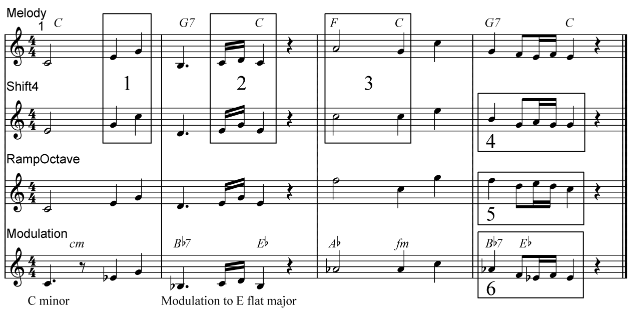

Figure 3: Transformations of Mozart Piano Sonata

The original melody appears in the top staff (transposed down an octave for clarity), and the harmonic progression is given above it. The next staff shows the same melody, shifted in register by four semitones. The third staff demonstrates a time varying shift in register using a linear ramp curve ending one octave higher. The final staff demonstrates the change of the underlying harmony, in this case a move to the key of C minor with a modulation taking place in bar 2 into the key of E flat major (the new progression is indicated above the stave). The numbers 1-6 placed above some staves identify key regions that are referenced in the following text.

Several observations are apparent in the comparison of the shifted melody in staff 2 to the original. Overall, the shift is successful in that the result sounds appropriate. The basic pattern of a skip between stable tone degrees in bar 1 is correctly shifted up in register. Note in region 1 that the pattern E-G in the melody (a third) was correctly transformed into the notes G-C (a fourth). This transformation cannot be accomplished through transposition alone, and illustrates Smart Harmony's unique ability.

Region 2 in measure 2 demonstrates a more subtle point. The two sixteenth notes in the melody serve to prolong the C, by embellishing it with the D. The first C in the melody is not a chord tone, whereas the D is (it is the 5th modal degree of the V_7 chord). The shift operation mapped this to E and G, where one would expect E and F. The reason for this is that both target notes for the second sixteenth note, F and G, are chord tones. Thus, with a shift of 4, our algorithm selected G, as it was closer to the transposed value. (Note however that the F would have been selected had we chosen a shift of 3 semitones).

Region 3 in measure 3 demonstrates how melodic shape can be affected by the shift in register. In this case, the downward movement from A to G in the melody is replaced by a repeated C note.

The shift result in region 4 may be considered appropriate, if the melody is embedded between other voices, or problematic, if the melody is the focus of the listener's attention. The first G in measure 4 of the melody creates a formational pattern with the preceding C in bar 3, where both notes are chord tones. However, in the shifted melody, the leap from E to B is problematic. The choice of jumping to chord tone B, while preserving the melodic shape, is troublesome from a functional point of view. As B is the leading tone, it would almost certainly be resolved within the style of this work by an upward step motion to C. Instead, it is resolved downwards to G. In this case, the most appropriate choice for shifting the G would be to the high D, which would alter the melodic shape but conform to the tonal system. The only other possible alternative would be to jump down to G, which would indeed conform to the tonal system, but would have an odd effect on the shape and flow of the melody.

It is interesting to note that the corresponding notes in the ramped-up melody (region 5) provide a very musical solution. Even though the melodic skip of a fourth down in the melody (C-G) is replaced with stepwise motion (G-F), the tonal resolution is perfect, whereby the 7th of the G_7 chord, F, resolves in a stepwise motion to E, then D, and finally to C. The corresponding location in the modulated melody (region 6) is somewhere in between. There is an uncharacteristic jump from C to A flat, the 7th of the dominant B7 chord, which should resolve to G. However the G here is skipped altogether. A musician in this case would most likely alter the melodic direction, having the notes of this last measure be: A flat, F, G (rather than E flat), F and E flat.

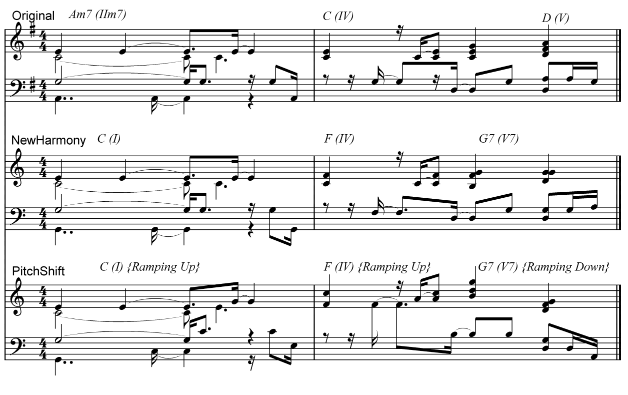

Figure 4: Piano Comp Example

In Figure 4, we modify a simple "comping riff" by first applying a new harmonic progression, and then applying a pitch shift (ramping upwards for 6 beats, then downward for two beats, peaking at the G7 chord with a 1 octave displacement). Note that the new progression defines a different harmonic rhythm: a pattern of 4-2-2 beats, compared to the original's 4-3-1. Also note that the original riff is intended to lead into the subsequent fragment (the last two sixteenth notes are non-chord tones), a feature that is properly preserved by the Smart Harmony mechanism.

Figure 3 illustrates the theoretical aspects of several Smart Harmony transformations, using an exemplar of 18th century music. Figure 4 shows how Smart Harmony has achieved our goal of enabling composers to quickly reshape musical materials in sensible and musically interesting ways, using a fragment of contemporary music.

5. FUTURE WORK

The Smart Harmony system provides a balance between theoretical correctness and composer flexibility – but it is by no means complete. It can still be improved on both of these counts, and this section presents our current ideas for improvements and extensions.

As implemented in Music Sketcher, all chord change operations (that do not utilize pitch-shifts) preserve both the register and harmonic function of the notes (through functional compatibility). However, this is not always desirable. There are times when relaxing the preservation of register would be more appropriate. In a bass line, for example, preserving register comes at the cost of changing the chord inversion, sometimes creating quite undesirable effects. At times, we have added pitch-shift modifiers specifically to undo the effects of register-preservation. This option should be exposed to the composer.

The harmony system itself should be made aware of chord inversions. The modal-degree computations will need to be rethought when inversions are permitted, but this should provide a musically sensible mechanism for the composer to control how the transformations operate.

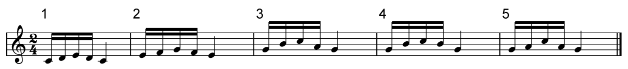

Figure 5: shifting a prolongation-pattern up in the scale

As noted, in shifting a melody in register one should maintain not only the function of each note but also the correct relationship between the stable and dynamic degrees, i.e. the melodic pattern. Consider the fragment in Figure 5, a melody in C major. Measure 1 consists of two connecting patterns: stable C to E; stable E to C. The dynamic tone, D, resolves first upwards and then downwards. Measure 2 demonstrates the next possible shift (a shift by a second is not possible: the distance between C and the next stable tone, E, is a third). Measure 2 is very similar to 1, and can be also obtained by diatonic transposition. However, the next possible shift in measure 3 demonstrates an important problem. Here, the stable tones, G and C, have two intervening dynamic tones: A and B, the 6th and 7th scale degrees. When shifting up as per measure 3, should the dynamic note F map to A or to B? Measure 3 has the musically correct answer: when the connective pattern ascends from G to C, one expects the leading tone to lead into C; when descending, one expects the 6th scale degree to resolve to 5. Measures 4 and 5 show solutions in which B and A (respectively) are selected: both are clearly less satisfactory. The selection of the correct note can only be done when considering the larger context of the enclosing melodic pattern, something that Smart Harmony does not yet handle.

In fact, this is just one instance of a more general problem: Smart Harmony does not examine melodic patterns. It works strictly "vertically" i.e. looking at a note only in its harmonic context, but not in its melodic context. The addition of a post-transformational "horizontal" analysis and correction should help Smart Harmony handle correct voice leading, ensure the correct resolution of dynamic tones, dealing with anticipation, embellishments, and correctly handle degrees 6 and 7 in both major and minor.

6. CONCLUSION

The Smart Harmony system discussed here provides an exceptional level of flexibility to the composer. The system allows rapid coarse-level changes such as the reshaping of material and combining musical materials from different contexts. This lets the composer quickly explore new musical territory, combining musical raw material in new ways. This is a significant step toward our goal of amplifying the composer's musical creativity.

However, the system is not yet complete. We have outlined a number of areas ripe for future exploration: voice-leading, proper representation of embellishments, appropriate handling of the minor key, and a more flexible mechanism for interpreting enharmonics, to name a few. We intend to continue to develop these ideas while remaining focused on our ultimate goal: utilizing technology to enhance composers' ability to be creative.

APPENDIX: PSEUDO-CODE

```
HarmonicContext::MIDIPitchFromChordPitch(cPitch)
{
       root = myTonality.SemiTonesFromTonic(myChordDegree);
       return myTonicMIDIPitch + root + cPitch.semiTones;
}
```

Figure 6: Computing the pitches of a Chord

```
HarmonicContext::MIDIPitchFromHarmonicPitch(hPitch, octave)
{
  if (hPitch.functionalModalDegree != 0) // hPitch IS a chord tone
  {
   s1= myTonality.SemiTonesFromTonic(myChordDegree);
   if (myChord.HasNoteWithModalDegree(hPitch.modalDegree))
    s2= myChord.semiTonesToDegree(hPitch.modalDegree);
   else
    s2= myChord.semiTonesToDegree(hPitch.functionalModalDegree);
   s2 += hPitch.alteration;
  }
  else                                    // hPitch IS NOT a chord Tone
  {
       steps = myChordDegree.scaleDegree + hPitch.modalDegree -1;
       if (steps > 7)
               steps -= 7; // Beware of possible octave wrapping
       alteration = myChordDegree.alteration + hPitch.alteration;
       s1 = myTonality.SemiTonesFromTonic(ScaleDegree(steps, alteration));
       s2 = 0;
  }
  return myTonicMIDIPitch + s1 + s2 + 12*octave;
}
```

Figure 7: Rendering a MIDI pitch from a Harmonic Pitch

```
HarmonicContext::MIDIPitchFromHarmonicPitchWithShift(hPitch, octave, semiTones)
{
  inMidiPitch=MIDIPitchFromHarmonicPitch(hPitch, octave);
  targetMidiPitch = inMidiPitch + semiTones; // The precisely transposed pitch.
  if (hPitch.functionalModalDegree != 0) // We have a chord tone here...
  {
    // For each chord-tone, compute an altered version of it,  Remember the closest one.
    For Each Chordal Pitch cPitch in myChord
    {
      // Construct testPitch: alteration from hPitch, modalDegree from cPitch.
      testPitch.modalDegree           = cPitch.modalDegree;
      testPitch.functionalModalDegree = cPitch.functionalModalDegree;
      testPitch.alteration = hPitch.alteration;
      testMIDIPitch = MIDIPitchFromHarmonicPitch(testPitch, octave);
      if testMIDIPitch closest to targetMIDIPitch, then outMIDIPitch = testMIDIPitch
    }
  }
  else   // Not a chord-tone; take a different tactic.
  {
    // For each non chord-tone,  compute a midiPitch and remember the closest one.
    For each modalDegree mDeg not present in myChord
    {
      testPitch.modalDegree           = mDeg;
      testPitch.functionalModalDegree = 0;
      testPitch.alteration            = hPitch.alteration;

      testMIDIPitch = MIDIPitchFromHarmonicPitch(testPitch, targetMidiPitch/12);
      if testMIDIPitch closest to targetMIDIPitch, then outMIDIPitch = testMIDIPitch
    }
    outMidiPitch = closestMidiPitch;
  }
 return outMIDIPitch;
}
```

Figure 8: Rendering a MIDI Pitch with Shift

```
HarmonicContext::HarmonicPitchFromMIDIPitch(midiPitch)
{
       for each chord pitch cPitch in myChord
       {
              if (MIDIPitchFromChordPitch(cPitch) == midiPitch)
                     return HarmonicPitch(midiPitch,
                                    cPitch.modalDegree, cPitch.functionalModalDegree, 0);
       }
       // myTonality is responsible for resolving which enharmonic name to use for midipitch
       inScaleDegree = myTonality.ScaleDegreeFromMIDIPitch(midiPitch);
       // Build theHarmonicPitch to return it.
       theHarmonicPitch->midiPitch   = midiPitch;
       theHarmonicPitch->modalDegree = (myScaleDegree.scaleDegree - myChordDegree.scaleDegree);
       theHarmonicPitch->functionalModalDegree = 0;
       // Find the ChordPitch, if any, that has the computed modalDegree
       for each ChordPitch cPitch in myChord
       {
              // Handle equivalence of, eg, 2nd and 9th in comparison
              if (cPitch.modalDegree == theHarmonicPitch->modalDegree ||
                  cPitch.modalDegree == theHarmonicPitch->modalDegree +7)
              {
                     theHarmonicPitch->modalDegree = cPitch.modalDegree;
                     theHarmonicPitch->functionalModalDegree = cPitch.functionalModalDegree;
                     break;
              }
       }
       if (theHarmonicPitch->functionalModalDegree != 0)
              theHarmonicPitch->alteration = distance(midiPitch, MIDIPitchFromChordPitch(cPitch);
       else
              theHarmonicPitch->alteration = myScaleDegree.alteration - myChordDegree.alteration;
       return theHarmonicPitch;
}
```

Figure 9: Rendering a MIDI Pitch with Shift

REFERENCES

Abrams, S., Oppenheim, D., Pazel, D., Wright, J., et. al. (1999). “Higher-Level Composition Control in Music Sketcher: Modifiers and Smart Harmony.” *Proceedings of the ICMC*, Beijing, China.

Levine, M, (1995). The Jazz Theory Book. Petaluma, CA: Scher Music Co.

Meyer, L., (1961) Emotion and Meaning in Music. University of Chicago Press.

Oppenheim, D. (1992). “Compositional Tools for Adding Expression to Music.” *Proceedings of the ICMC,* San Jose, California.

Oppenheim, D. (1996). “DMIX—A Multi Faceted Environment for Composing and Performing Computer.” Computers and Mathematics with Applications, 32(1): 117-135.

Pachet, F. (1993). “An Object-Oriented Representation of Pitch-Classes, Intervals, Scales and Chord: The basic MusES.” (revised and extended version), LAFORIA, Internal Report 93/38, November 1993.

Pazel, D., Abrams, S., Fuhrer, R., Oppenheim, D., Wright, J. (2000). “A Distributed Interactive Music Application using Harmonic Constraint.” *Proceedings of the ICMC*, Berlin, Germany.

Ratner, Leonard G. (1985) Classic Music : Expression, Form and Style. Wadsworth Pub Co.

Sadai, Y. (1980) Harmony in its Systemic and Phnomenological Aspects. Jerusalem, Israel: Yanetz Press.

Schenker, H., (1969) "Five Graphic Music Analyses” Dover Publications, Inc.

Wright, J., Oppenheim, D., Jameson, D., Pazel D., Fuhrer R. (1997). “CyberBand: A ‘Hands-On’ Music Composition Program.” *Proceedings of the ICMC,* Thessaloniki, Greece.